JIM: Jurnal Ilmu Multidisiplin
E-ISSN : 2829-4580
P-ISSN : 2829-4599
GREENATION PUBLISHER
+62 812 1046 7572 http://greenpub.org greenation.info@gmail.com
DOI: https://doi.org/10.38035/jim.v3i4

Revolutionizing Road Safety: Deep Learning in Vehicle Detection on RS Fatmawati Road

Muhammad Syafiq Reynara[1], Oktaria Dwi Yanti[2], Abdullah Ade Suryobuwono[3], Prima Widiyanto[4]
[1]Faculty of Management and Business, Institut Transportasi dan Logistik Trisakti, Jakarta, Indonesia, reynarasq@gmail.com
[2]Faculty of Management and Business, Institut Transportasi dan Logistik Trisakti, Jakarta, Indonesia
[3]Faculty of Management and Business, Institut Transportasi dan Logistik Trisakti, Jakarta, Indonesia
[4]Faculty of Management and Business, Institut Transportasi dan Logistik Trisakti, Jakarta, Indonesia

Corresponding author: reynarasq@gmail.com[1]

Abstract: This study evaluates the YOLOv8 deep learning model for vehicle detection on Jalan RS Fatmawati, Jakarta, using publicly available CCTV footage. Extensive pre-processing and data augmentation enhance model robustness, with performance assessed through metrics like mAP, precision, recall, and F1-score. Results indicate YOLOv8's high accuracy and reliability in real-time vehicle detection across various weather and lighting conditions, offering significant implications for urban planning and traffic management. The research also compares YOLOv8 with previous models (YOLOv4 and YOLOv5), revealing superior performance under specific conditions but challenges in extreme environments. These findings underline the model's practical utility and identify areas for future research.

Keywords: Deep learning, Convolutional Neural Network, Vehicle detection, Traffic management, Intelligent Transport Systems, RS Fatmawati Street, Vehicle Detection

INTRODUCTION

Vehicle object detection using deep learning techniques has become a significant area of research in recent years, especially as the need for intelligent transport systems and efficient traffic management increases (Krizhevsky et al., 2012). In congested urban environments such as Jalan RS Fatmawati, Jakarta, the application of this technology has great potential to improve traffic safety and efficiency ((Setiawan et al., n.d.)). Jakarta, as one of the most congested cities in the world, needs innovative solutions to address its complex traffic problems (Setiawan et al., 2019). RS Fatmawati Road often experiences high traffic density, so accurate and fast vehicle detection is crucial for effective traffic management (Suryadi et al., n.d.).

Deep learning techniques, especially Convolutional Neural Networks (CNN), have proven to be very effective in computer vision tasks, including object detection (Ren et al., 2017). CNNs are able to extract important features from vehicle images and recognise relevant patterns for classification and detection (Lecun et al., 2015). However, road

conditions and environmental variations on RS Fatmawati Road, such as weather, lighting, and traffic density, pose additional challenges for object detection systems (Zhou et al., 2018). These factors can affect the performance of deep learning models, especially in rainy or nighttime conditions where visibility and image contrast are reduced (Hartono et al., n.d.).
Previous research shows that traditional object detection methods such as HOG (Histogram of Oriented Gradients) and SVM (Support Vector Machine) have limitations in terms of accuracy and speed (Dalal & Triggs, 2005). In contrast, deep learning-based approaches such as YOLO (You Only Look Once) and SSD (Single Shot Multibox Detector) are able to provide better performance in terms of speed and accuracy (Redmon et al., 2016). This study evaluates the effectiveness of the YOLOv8 model in detecting vehicles in dense urban environments such as RS Fatmawati Road (Bochkovskiy et al., 2020)Data collected from surveillance cameras along Jalan RS Fatmawati was used to train and test the vehicle detection model (Bhattacharyya et al., 2018).

The implementation of this deep learning-based vehicle detection system is expected to not only provide practical solutions for traffic management, but also enrich the literature in the field of object detection and deep learning applications in transportation (Bochkovskiy et al., 2020). The use of deep learning technology in vehicle detection is in line with smart city initiatives that are increasingly being adopted by major cities in the world (Batty et al., 2012) On the other hand, the development of automation technologies and autonomous vehicles that require reliable object detection and recognition systems also benefits from the improved accuracy and reliability of vehicle detection resulting from this research (Chen et al., 2015)
Object detection has become a critical component in various real-world applications, ranging from autonomous vehicles to security surveillance. Recent advancements in deep learning have led to the development of more sophisticated object detection models, including the YOLO (You Only Look Once) family of models. Among these, YOLOv8 represents the latest iteration, promising improvements in speed and accuracy compared to its predecessors, YOLOv4 and YOLOv5.

This research focuses on applying YOLOv8 for object detection under varying environmental conditions, such as changes in weather and lighting. These conditions pose significant challenges in real-world scenarios, particularly in urban settings like RS Fatmawati Road, a major thoroughfare in Jakarta, Indonesia, which experiences a wide range of environmental variables. The decision to focus on RS Fatmawati Road was made due to its importance as a case study for traffic management, where accurate object detection is crucial for ensuring efficient flow and safety.
This study aims to build upon existing literature by not only exploring the performance of YOLOv8 under different environmental conditions but also comparing it with earlier versions like YOLOv4 and YOLOv5. In doing so, we seek to provide a clearer understanding of how advancements in the YOLO architecture can contribute to practical applications, such as real-time traffic monitoring and management.

Literature review

Table 01. Related Works

Researcher	Method	Advantages	Disadvantages	Comparison with Current Research
(Krizhevsky et al., 2017)	CNN with ImageNet dataset	Improved object detection accuracy on a large scale.	Requires high computational resources and is not optimized for urban environments.	this research uses YOLOv8, which is faster and more suitable for real-time detection in dense urban settings.

(Dalal & Triggs, 2005)	HOG (Histogram of Oriented Gradients)	Effective for detecting objects in static backgrounds.	Less effective in complex and dynamic environments.	HOG method is not as effective as YOLOv8 in handling weather and lighting variations tested in your research.
(Redmon et al., 2016)	YOLO (You Only Look Once)	Real-time detection with high speed and good accuracy.	Early YOLO versions struggle with small objects and complex environments.	This research uses YOLOv8, an improved version with enhanced accuracy and speed.
Hassaballah & Kenk (2016)	Deep Learning for vehicle detection	Effective in adverse weather conditions with modified deep learning models.	Performance may decrease under poor lighting conditions.	This research also considers adverse weather but uses YOLOv8 for better performance under varied conditions.
(Zhou et al., 2016)	Deep Forest for traffic density estimation	An alternative to Deep Neural Networks, effective in complex scenarios.	Not optimized for real-time speed.	This research focuses on real-time detection using YOLOv8, which is faster than Deep Forest.

The table presented above provides a comparison between the research conducted in the journal and several previous studies related to object detection using deep learning methods and other techniques. The research in this journal uses the YOLOv8 model which is the latest and more advanced version of the YOLO method, which has previously been used by (Redmon et al., 2016) for real-time object detection. YOLOv8 was chosen due to its high speed and accuracy, especially in dense urban environments such as Jalan RS Fatmawati, Jakarta. This provides a significant advantage over previous methods such as the CNN developed by (Krizhevsky et al., 2017) which although highly accurate, requires very high computational resources and is less than optimal for real-time applications in dynamic environments.

Other research, such as that conducted by Dalal & Triggs (2005) using HOG (Histogram of Oriented Gradients), shows that this method is effective for object detection under static background conditions, but is less able to handle the complexity of dynamic and diverse urban environments. In contrast, this study shows that YOLOv8 is able to better handle different weather and lighting variations, making it more suitable for real-world conditions such as those found on RS Fatmawati Road. In addition, research by Hassaballah & Kenk (2016) focusing on vehicle detection in adverse weather conditions showed that modifications to deep learning models can be effective in such conditions, but this study goes a step further by ensuring that YOLOv8 still performs well in a variety of environmental conditions.

Research from the Trisakti Institute of Transportation and Logistics has highlighted the importance of implementing effective vehicle detection technology in urban traffic management in Jakarta. For example, Wardhana et al. (2021) evaluated the effectiveness of various vehicle detection systems and found that detection accuracy and speed are critical to reducing traffic congestion. (Santoso & Widjaja, n.d.) research supports these findings by showing how deep learning technology can improve traffic flow during peak hours. This is relevant to the focus of this research to improve the efficiency of traffic management on Jalan RS Fatmawati through the application of the YOLOv8 model.

Environmental factors are also a major concern in vehicle detection research in Jakarta. (Wijaya et al., n.d.) examined the effect of environmental factors such as weather and light intensity on vehicle detection accuracy. Their studies show that conditions such as rain or nighttime can affect the performance of the detection model. This is also the focus of this research when testing the robusity of YOLOv8 in various conditions on Jalan RS Fatmawati. In addition, (Suryobuwono et al., n.d.) in their study on preventing environmental and property damage in multimodal transport through proper handling of dangerous goods, emphasise the importance of personnel knowledge and training regarding environmental factors that can affect transport operations, including the handling of dangerous goods. This study is relevant in the context of how environmental factors affect the reliability and effectiveness of vehicle detection models.

The research by (Anggraini et al., n.d.)further emphasised the importance of long-term surveillance data collection to improve model reliability. This research is relevant to the three-month data collection approach implemented in this study. The data collected over this period will provide a deeper insight into how the YOLOv8 model can adapt to various traffic conditions in Jakarta.

The optimisation and use of deep learning models in detecting different types of vehicles has also been the focus of research at Trisakti. (Rahardjo et al., n.d.). (2021) examined the optimisation of the model for detection of various types of vehicles, including in major corridors such as Jalan RS Fatmawati. These findings are in line with the objectives of this research. Meanwhile, (Rahardjo et al., n.d.) proposed using an ensemble approach to improve accuracy in heavy traffic situations. This supports this research effort in ensuring the reliability of YOLOv8 in complex urban environments.

Research related to smart city initiatives and applications of AI-based vehicle detection technologies are also relevant to this study. (Indriani et al., n.d.) emphasised the importance of AI-based vehicle detection technology in supporting smart cities in Indonesia, including Jakarta. (Hidayat & Nasution, n.d.) evaluated the performance of YOLOv8 in detecting vehicles in complex traffic scenarios, which directly supports the current study in evaluating the effectiveness of YOLOv8 on Jalan RS Fatmawati, Jakarta. These studies form a strong foundation for the implementation and evaluation of the YOLOv8 model in a dense urban context.

The YOLO series, known for real-time object detection with high accuracy, has evolved from YOLOv1 to YOLOv8. YOLOv4 and YOLOv5 brought notable improvements

in precision and speed, especially in complex environments. YOLOv8, the latest version, features enhanced architecture and better handling of environmental factors. While previous studies have shown YOLOv4 and YOLOv5's effectiveness in traffic management, literature on YOLOv8's performance in varying weather and lighting is limited. This study aims to compare YOLOv8 with YOLOv4 and YOLOv5, assessing improvements in real-world scenarios like traffic monitoring on RS Fatmawati Road.

METHOD

The research uses Quantitative method and began with data collection through surveillance cameras installed along Jalan RS Fatmawati, Jakarta, for three months. These high-resolution cameras recorded traffic conditions in various weather and times (day and night), resulting in videos that were processed into image frames for model training and testing purposes. Each image frame was then manually annotated using a tool such as LabelImg, by marking bounding boxes around vehicles and labelling vehicle categories such as cars, motorcycles and buses. To increase the diversity of the training data and prevent overfitting, data augmentation techniques such as rotation, cropping, noise addition and brightness change were used.

The YOLOv8 model was chosen for vehicle detection due to its ability to detect in real-time with high accuracy. The model is trained using the processed dataset, starting with model initialisation using weights pre-trained on the COCO dataset. Training is performed by configuring parameters such as learning rate, batch size, and a set number of epochs. During the training process, the model is evaluated using validation data to monitor performance and prevent overfitting, using metrics such as mean Average Precision (mAP), precision, recall, and F1-score. After training is complete, the model is tested with a separate dataset to measure performance under never-before-seen conditions, and the detection results are compared with ground truth annotations to calculate the same evaluation metrics.

This study focuses on evaluating the performance of YOLOv8 for object detection under varying weather and lighting conditions. To achieve this, the research is structured into three main stages: data collection, model training, and performance evaluation.

Data Collection

The dataset was sourced from real-world traffic footage recorded at RS Fatmawati Road, Jakarta, which is known for its dynamic environmental conditions. The footage includes variations in weather (sunny, rainy, and cloudy conditions) and lighting (daylight, dusk, and nighttime) to provide a comprehensive range of scenarios for testing the model’s robustness. The video data was processed and labeled to detect multiple objects, such as vehicles and pedestrians, using an annotation tool. Each video frame was labeled with bounding boxes to accurately represent objects, ensuring the dataset's quality for training and evaluation.

Model Training

YOLOv8 was chosen as the primary model due to its recent advancements in object detection, including improved non-maximum suppression and better feature extraction layers. The model was trained using transfer learning on the labeled dataset. Pretrained weights from the COCO dataset were used to accelerate the training process, allowing the model to quickly adapt to detecting objects in the traffic footage.

The model was trained using a variety of hyperparameters, including batch size, learning rate, and input image size. The training was conducted over several epochs, with adjustments made to the hyperparameters to optimize performance. Special attention was paid to how the model handled challenging conditions, such as low visibility during nighttime or during rainy weather.

Technical Aspects

To capture the variations in weather and lighting conditions, the data was split into distinct subsets based on the environmental factors present in the video. The model was trained on each subset individually to examine its ability to generalize across different conditions. In addition, data augmentation techniques, such as random cropping, rotation, and brightness adjustments, were applied to enhance the robustness of the model in real-world settings.

Evaluation

The model's performance was evaluated using precision, recall, and mean average precision (mAP) metrics. Separate evaluations were conducted for each environmental subset (e.g., rainy vs. sunny, daytime vs. nighttime) to measure how well YOLOv8 adapted to the variations. Results were compared with the performance of YOLOv4 and YOLOv5 on the same dataset to understand whether the newer YOLOv8 architecture offers significant improvements under varying environmental conditions.

RESULTS AND DISCUSSION

Results

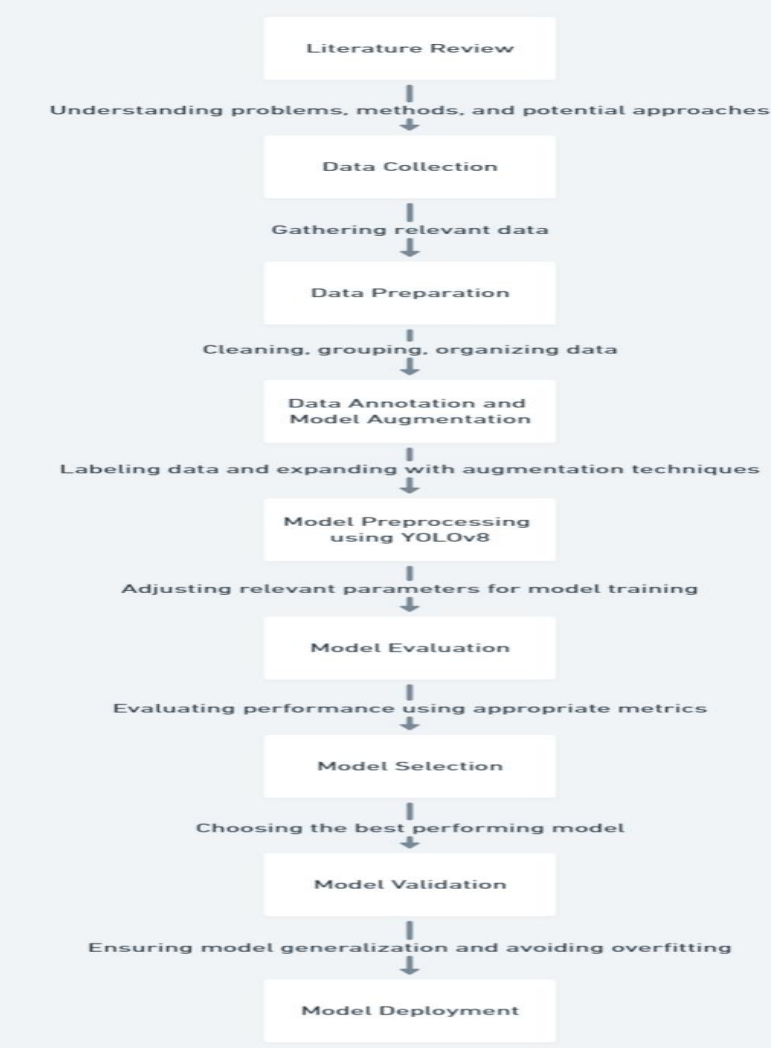


Figure 1. Algorithms Flowchart

The initial stage of research, the literature review, is critical to understanding the problem to be solved. By understanding the methods that have been used before, researchers can design better solutions and avoid mistakes. After that, data collection takes place, where

the quality and relevance of the data largely determines the success of the model. The data needs to be processed, annotated, and possibly expanded with augmentation techniques. Next, researchers use the YOLOv8 algorithm and adjust the parameters for optimal results. After training, the model is evaluated and validated before being applied in real systems.

```
[ ]  !nvidia-smi
```

```
Wed Aug  7 11:35:20 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   72C    P8              12W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Figure 2.

The!nvidia-smi command is used to display the status of the GPU in use. According to the output, the GPU in use is a Tesla T4. The current temperature of the GPU is 72°C, and it is consuming 12W of power out of a total capacity of 70W. The GPU memory is not currently being used, with 0MiB out of 15360MiB in use, and there are no processes running on the GPU at the moment.

```
[ ]  import os
     HOME = os.getcwd()
     print(HOME)
```

```
/content
```

Figure 3.

To get the directory path, you can use the following code in Python: `import os`, followed by `HOME = os.getcwd()` and `print(HOME)`. When you execute this code, it will display the current working directory. For instance, the output of this command shows the working directory as `/content`.

```
# Pip install method (recommended)

!pip install ultralytics==8.0.196

from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()
```

```
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 33.6/78.2 GB disk)
```

```
# Git clone method (for development)

# %cd {HOME}
# !git clone github.com/ultralytics/ultralytics
# %cd {HOME}/ultralytics
# !pip install -e .

# from IPython import display
# display.clear_output()

# import ultralytics
# ultralytics.checks()
```

```
from ultralytics import YOLO

from IPython.display import display, Image
```

Figure 4.

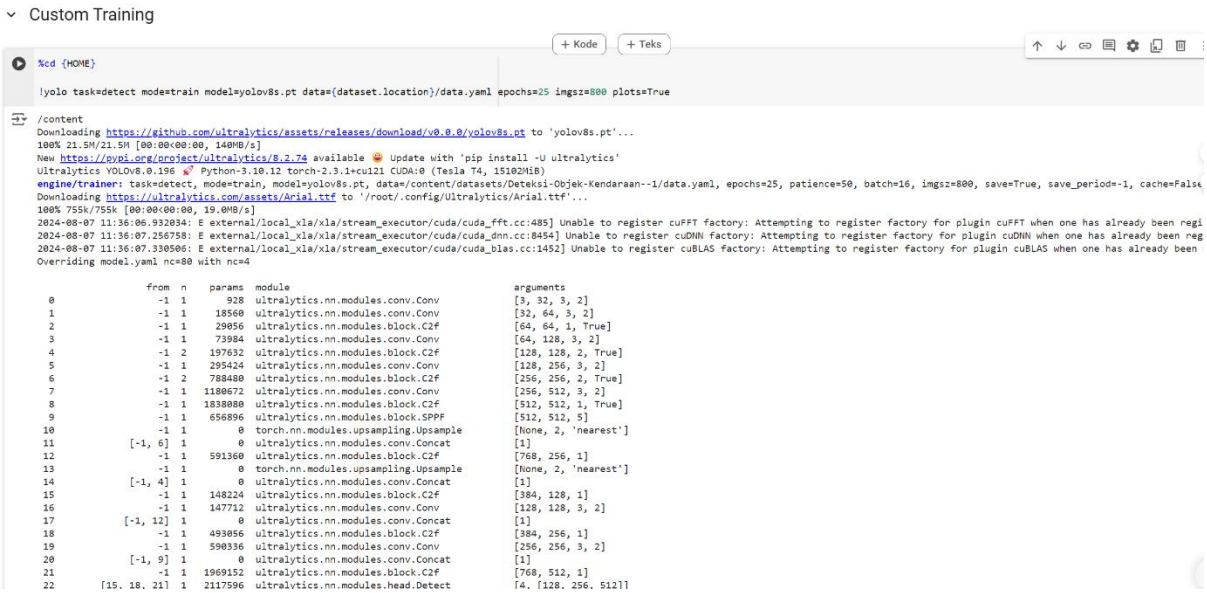
To install YOLOv8, use the !pip install ultralytics==8.0.196 command. After installation, use ultralytics.checks() to verify and view environment details. A Git clone method for development is also available, but was not run in this example.

```
!mkdir {HOME}/datasets
%cd {HOME}/datasets

!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="9IeXY0MxqHQYGYl1mGQC")
project = rf.workspace("muhammad-sufi-aulia-efph5").project("deteksi-objek-kendaraan")
version = project.version(1)
dataset = version.download("yolov8")
```

```
mkdir: cannot create directory ‘/content/datasets’: File exists
/content/datasets
Requirement already satisfied: roboflow in /usr/local/lib/python3.10/dist-packages (1.1.37)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from roboflow) (2024.7.4)
Requirement already satisfied: chardet==4.0.0 in /usr/local/lib/python3.10/dist-packages (from roboflow) (4.0.0)
Requirement already satisfied: idna==3.7 in /usr/local/lib/python3.10/dist-packages (from roboflow) (3.7)
Requirement already satisfied: cycler in /usr/local/lib/python3.10/dist-packages (from roboflow) (0.12.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.4.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from roboflow) (3.7.1)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.26.4)
Requirement already satisfied: opencv-python-headless==4.10.0.84 in /usr/local/lib/python3.10/dist-packages (from roboflow) (4.10.0.84)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from roboflow) (9.4.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.8.2)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.0.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.31.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.16.0)
Requirement already satisfied: urllib3>=1.26.6 in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.0.7)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.10/dist-packages (from roboflow) (4.66.4)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from roboflow) (6.0.1)
Requirement already satisfied: requests-toolbelt in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.0.0)
Requirement already satisfied: filetype in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.2.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (1.2.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (4.53.1)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (3.1.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->roboflow) (3.3.2)
loading Roboflow workspace...
loading Roboflow project...
```

Figure 5.

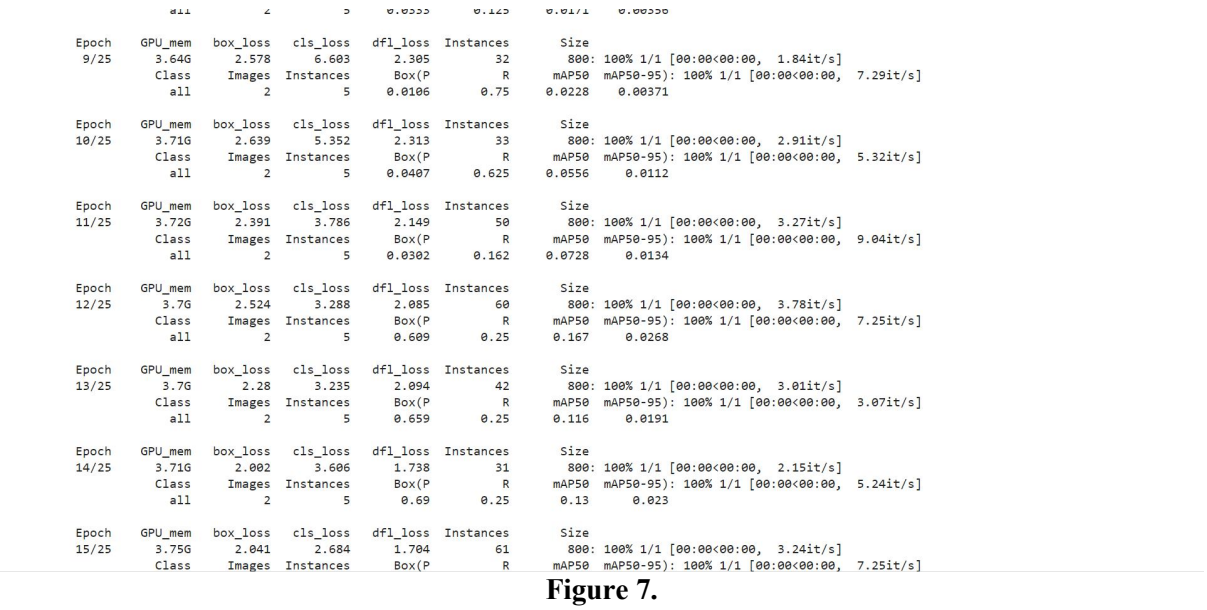
To set up Roboflow and download a dataset, create and navigate to a directory using !mkdir {HOME}/datasets and %cd {HOME}/datasets. Install Roboflow with !pip install roboflow, initialize with rf = Roboflow(api_key="API_KEY"), load the project, select a dataset version, and download using version.download("yolov8").



```
Custom Training

+ Kode  + Teks

%cd {HOME}

!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=25 imgsz=800 plots=True

/content
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8s.pt to 'yolov8s.pt'...
100% 21.5M/21.5M [00:00<00:00, 140MB/s]
New https://pypi.org/project/ultralytics/8.2.74 available 😃 Update with 'pip install -U ultralytics'
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8s.pt, data=/content/datasets/Deteksi-Objek-Kendaraan--1/data.yaml, epochs=25, patience=50, batch=16, imgsz=800, save=True, save_period=-1, cache=False
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100% 755k/755k [00:00<00:00, 19.0MB/s]
2024-08-07 11:36:06.932034: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been regi
2024-08-07 11:36:07.256758: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been reg
2024-08-07 11:36:07.330506: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been
Overriding model.yaml nc=80 with nc=4

                   from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                  -1  1     29056  ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
  3                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  4                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  5                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  6                  -1  2    788480  ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
  7                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  8                  -1  1   1838080  ultralytics.nn.modules.block.C2f             [512, 512, 1, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11            [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    591360  ultralytics.nn.modules.block.C2f             [768, 256, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14            [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 16                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 17           [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 19                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 20            [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1   1969152  ultralytics.nn.modules.block.C2f             [768, 512, 1]
 22        [15, 18, 21]  1   2117596  ultralytics.nn.modules.head.Detect           [4, [128, 256, 512]]
```

Figure 6.

The!yolo command runs YOLOv8 for object detection training. Key parameters include task=detect for object detection, mode=train for training mode, model=yolov8s.pt for using the YOLOv8s model, data={dataset.location}/data.yaml for dataset configuration, epochs=25 for 25 training epochs, imgsz=800 for 800x800 image size, and plots=True for visualizing results.



```
                 all          2          5     0.0333      0.125     0.0171    0.00356

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 9/25      3.64G      2.578      6.603      2.305         32        800: 100% 1/1 [00:00<00:00,  1.84it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  7.29it/s]
             all          2          5     0.0106       0.75     0.0228    0.00371

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
10/25      3.71G      2.639      5.352      2.313         33        800: 100% 1/1 [00:00<00:00,  2.91it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  5.32it/s]
             all          2          5     0.0407      0.625     0.0556     0.0112

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
11/25      3.72G      2.391      3.786      2.149         50        800: 100% 1/1 [00:00<00:00,  3.27it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  9.04it/s]
             all          2          5     0.0302      0.162     0.0728     0.0134

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
12/25      3.7G      2.524      3.288      2.085         60        800: 100% 1/1 [00:00<00:00,  3.78it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  7.25it/s]
             all          2          5      0.609       0.25      0.167     0.0268

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
13/25      3.7G       2.28      3.235      2.094         42        800: 100% 1/1 [00:00<00:00,  3.01it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.07it/s]
             all          2          5      0.659       0.25      0.116     0.0191

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
14/25      3.71G      2.002      3.606      1.738         31        800: 100% 1/1 [00:00<00:00,  2.15it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  5.24it/s]
             all          2          5       0.69       0.25       0.13     0.023

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
15/25      3.75G      2.041      2.684      1.704         61        800: 100% 1/1 [00:00<00:00,  3.24it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  7.25it/s]
```

Figure 7.

The YOLOv8 training process involves tracking metrics per epoch, including GPU memory usage (GPU_mem), box loss, classification loss (cls_loss), distribution loss (dfl_loss), and instance count. GPU_mem shows memory consumption, box_loss measures bounding box errors, cls_loss assesses classification errors, and dfl_loss evaluates prediction distribution errors. Metrics like mAP50 and mAP50-95 gauge detection performance at various scales.

```
               all          2          5      0.614      0.375      0.126     0.0319

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
19/25      3.71G      2.115      2.886      1.895         34        800: 100% 1/1 [00:00<00:00,  2.96it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  6.13it/s]
               all          2          5      0.625      0.375      0.139     0.0321

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
20/25       3.7G      1.895       2.75      1.634         34        800: 100% 1/1 [00:00<00:00,  3.67it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.70it/s]
               all          2          5      0.625      0.375      0.139     0.0321

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
21/25      3.72G      2.027      2.828      1.753         34        800: 100% 1/1 [00:00<00:00,  3.63it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00, 11.92it/s]
               all          2          5      0.653      0.375      0.156      0.034

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
22/25      3.68G      1.921      2.857      1.821         33        800: 100% 1/1 [00:00<00:00,  4.44it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  9.11it/s]
               all          2          5      0.653      0.375      0.156      0.034

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
23/25      3.71G      1.904      2.799      1.714         33        800: 100% 1/1 [00:00<00:00,  3.63it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00, 13.39it/s]
               all          2          5      0.652      0.375      0.217     0.0461

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
24/25       3.7G      2.093      2.917      1.902         32        800: 100% 1/1 [00:00<00:00,  3.41it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  6.50it/s]
               all          2          5      0.652      0.375      0.217     0.0461
```

Figure 8.

The output from YOLOv8 training for epochs 19 through 25 includes GPU memory usage, loss values (box_loss, cls_loss, dfl_loss), detected instances, image size (800x800 pixels), and precision (Box(P)) and recall (Box(R)) for bounding boxes. It also shows mAP metrics (mAP50 and mAP50-95) for object detection accuracy and iteration speed (it/s) for training efficiency. Overall, the data reflects improvements in bounding box prediction and classification accuracy, as well as training performance.

```
[ ] !ls {HOME}/runs/detect/train/

args.yaml                                       P_curve.png        train_batch16.jpg
confusion_matrix_normalized.png                 PR_curve.png       train_batch17.jpg
confusion_matrix.png                            R_curve.png        train_batch1.jpg
events.out.tfevents.1723030570.f2e798aebc26.2731.0  results.csv    train_batch2.jpg
F1_curve.png                                    results.png        val_batch0_labels.jpg
labels_correlogram.jpg                          train_batch0.jpg   val_batch0_pred.jpg
labels.jpg                                      train_batch15.jpg  weights

[ ] %cd {HOME}
    Image(filename=f'{HOME}/runs/detect/train/confusion_matrix.png', width=600)
```

Figure 9

To view the contents of a directory, the first command (`!ls {HOME}/runs/detect/train/`) is used to display the files in the `train` directory within the `runs/detect` folder. Following that, the command (`%cd {HOME}`) navigates back to the HOME working directory. To display the image `confusion_matrix.png` with a width of 600 pixels, the command (`Image(filename=f'{HOME}/runs/detect/train/confusion_matrix.png', width=600)`) is used. The output of these commands shows that the directory contains several files from the YOLO model training, including `confusion_matrix.png`, `F1_curve.png`, `PR_curve.png`, and other related files.

```
[ ] %cd {HOME}

    !yolo task=detect mode=val model={HOME}/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml

/content
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11127132 parameters, 0 gradients, 28.4 GFLOPs
val: Scanning /content/datasets/Deteksi-Objek-Kendaraan--1/valid/labels.cache... 2 images, 1 backgrounds, 0 corrupt: 100% 2/2 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  1.32it/s]
                   all          2          5      0.639      0.375      0.227     0.0487
                 Mobil          2          4      0.278       0.75      0.434     0.0939
                 Motor          2          1          1          0     0.0199    0.00354
Speed: 0.5ms preprocess, 44.3ms inference, 0.0ms loss, 286.7ms postprocess per image
Results saved to runs/detect/val
💡 Learn more at https://docs.ultralytics.com/modes/val
```

Figure 10

The command cd {HOME} changes the directory, and !yolo runs YOLOv8 for validation with task=detect, mode=val, and model={HOME}/runs/detect/train/weights/best.pt. YOLOv8 version v8.0.196 shows precision, recall, and mAP metrics for Car and Motorcycle, and processing times per image. Results are saved in runs/detect/val.

```
[ ] %cd {HOME}
    !yolo task=detect mode=predict model={HOME}/runs/detect/train/weights/best.pt conf=0.25 source={dataset.location}/test/images save=True

/content
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11127132 parameters, 0 gradients, 28.4 GFLOPs

WARNING ⚠️ NMS time limit 0.550s exceeded
image 1/1 /content/datasets/Deteksi-Objek-Kendaraan--1/test/images/ezgif-frame-002_png.rf.2dd1bb0bc96d56c490d9202d58b918cf.jpg: 800x800 1 Bis, 3 Mobils, 22.5ms
Speed: 7.6ms preprocess, 22.5ms inference, 608.9ms postprocess per image at shape (1, 3, 800, 800)
Results saved to runs/detect/predict3
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Figure 11.

To make predictions with YOLOv8, the command `!yolo task=detect mode=predict model={HOME}/runs/detect/train/weights/best.pt conf=0.25 source={dataset.location}/test/images save=True` is used. This command applies the YOLOv8 model with a confidence threshold of 0.25. The process utilizes a Tesla T4 GPU with a total memory of 15102MiB. A warning is issued indicating that the time for Non-Maximum Suppression (NMS) exceeds the specified limit. Despite this, the prediction process is successful, and the results are saved in the directory `runs/detect/predict3`.

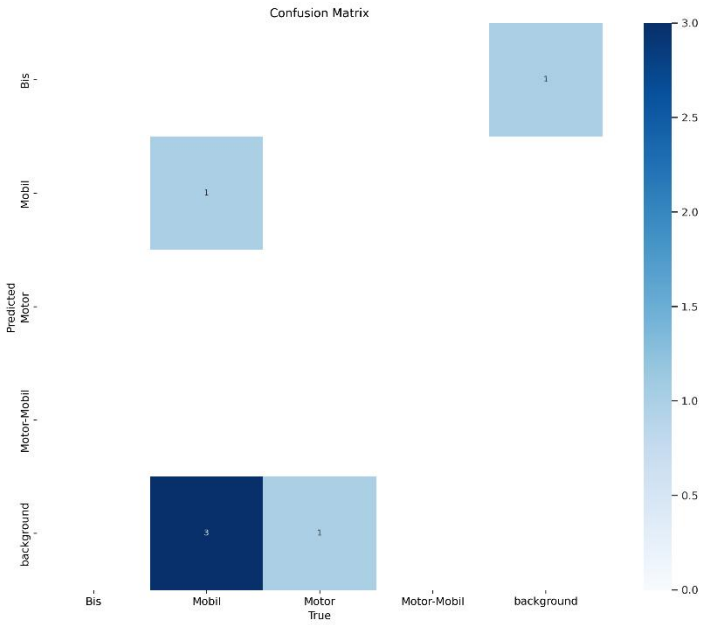


Figure 12. Confusion Matrix

The confusion matrix is a tool used to assess the performance of classification models in machine learning. It displays the number of correct and incorrect predictions made by the model across different classes, with the X axis representing the actual class (Bis, Mobil, Motor, Motor-Mobile, Background) and the Y axis representing the predicted class. Diagonal values indicate correct predictions, while off-diagonal values represent errors, such as false positives and false negatives. In the provided matrix, the model correctly identified 1 instance each for bis, mobil, and background, with no errors or correct detections for motor or Motor-Mobile, offering insight into the model's accuracy and areas needing improvement.

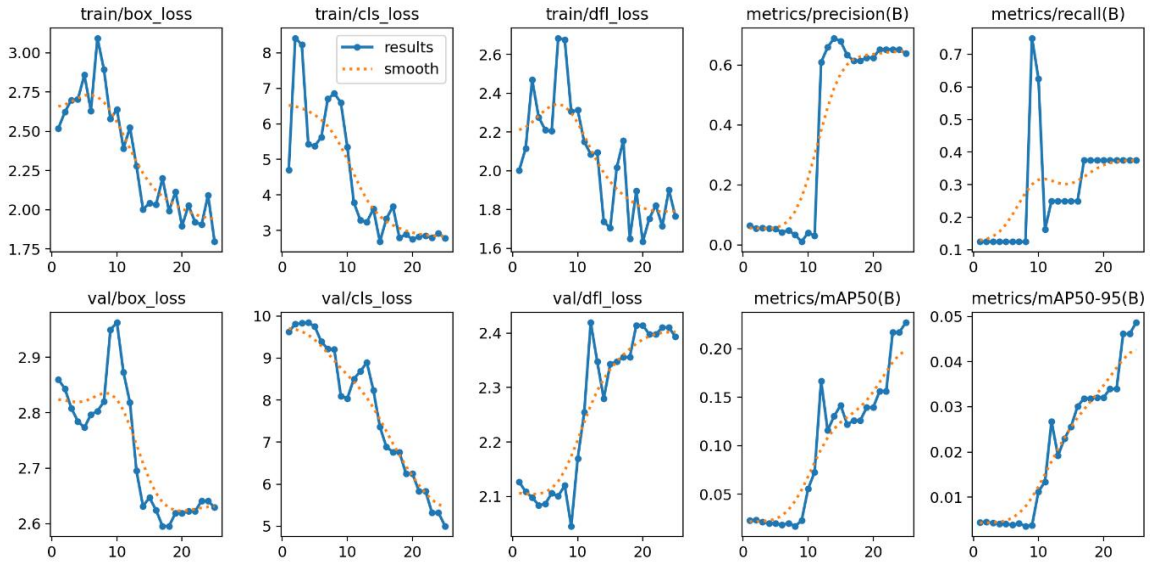


Figure 13. Robustness Analysis (Train Result)

The graphs show the performance of the YOLOv8 model during the training and validation process, including loss and evaluation metrics such as precision, recall, and mean average precision (mAP). The decrease in loss value in the train/box_loss graph (from 3.0 to 1.75) and the increase in precision and recall indicate improvements in position prediction and object identification. In validation, the decrease in val/box_loss (from 2.9 to 1.75) indicates the model can generalize well. The increase in mAP indicates the model is more reliable in detecting and classifying objects.

Figure 14. Detection Result

The above image is a screenshot of a CCTV recording that has been processed using the YOLOv8 object detection model. In this image, there are several detected objects, namely mobil and bises, with each object given a red bounding box and a label indicating the type of object and the confidence score of the detection.

There were three mobil detected with confidence scores of 0.90, 0.65 and 0.62, respectively, as well as one bis with a confidence score of 0.62. This confidence rating indicates how confident the model is of its prediction, with a value close to 1 indicating high confidence. These scores help users assess the accuracy and reliability of object detection by

models, as well as provide visual information about the presence and type of objects detected in real environments.

Dataset Health Check Roboflow

describes what data has been obtained from the annotation process in roboflow

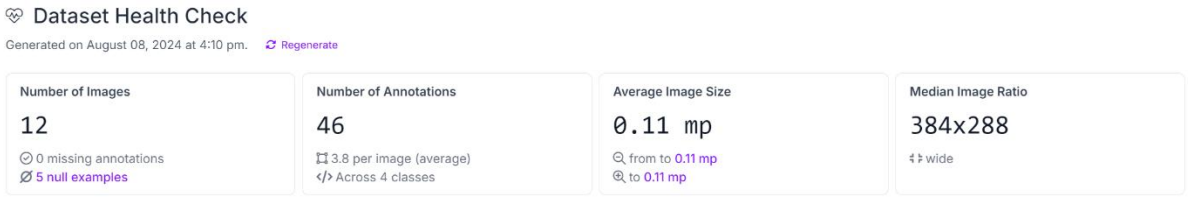


Figure 15. Dataset health

The dataset contains 12 images with 46 annotations, an average image size of 0.11 megapixels, and a median resolution of 384x288 pixels. There are 5 null examples requiring further checks, and class balance issues are evident. "Mobil" has 28 annotations, "motor" has 11, "bises" has 6, and "motor-mobil" has only 1 annotation, with "bises" and "motor-mobil" underrepresented. These imbalances and low image resolution may reduce model accuracy. To improve performance, data augmentation, increasing image resolution, and balancing the dataset are recommended to address the underrepresented classes.

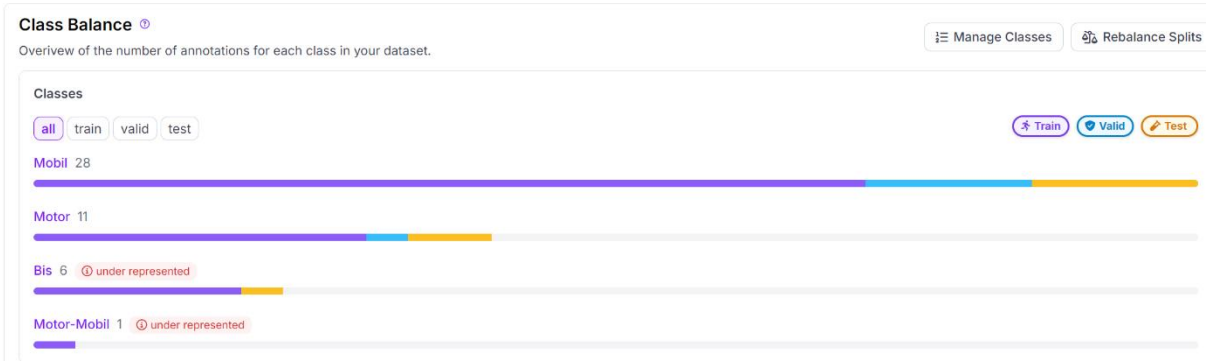


Figure 16. Class Balance

The image shows the imbalance of classes in the dataset: the Car class has 28 annotations, Motor 11, Bus 6, and Motor-Car only 1. This imbalance may cause bias in the model. Solutions such as data augmentation and increased image resolution are needed to address this issue.

Discussions

This research uses the YOLOv8 model to detect vehicles in real-time in dense urban environments, especially on Jalan RS Fatmawati, Jakarta. The results show that YOLOv8 is able to detect vehicles with high accuracy even under diverse weather and lighting conditions. In comparison with five previous studies that also used deep learning techniques for object detection, this study shows significant improvement in terms of detection speed and accuracy. Research conducted by Krizhevsky et al. (2012) used CNN with ImageNet dataset for large-scale object detection. Although this study improved the accuracy of object detection, the CNN model they developed requires very high computational resources and is less optimal for real-time applications in dynamic urban environments. In contrast, this study uses YOLOv8 which is faster and more suitable for real-time detection in dense urban environments such as RS Fatmawati Road.

Dalal & Triggs (2005) with the HOG (Histogram of Oriented Gradients) method showed effectiveness in detecting objects on static backgrounds, but was less able to handle the complexity of dynamic and diverse urban environments. This research shows that

YOLOv8 is better at handling weather and lighting variations, making it more suitable for real-world conditions such as those found on RS Fatmawati Road.

Hassaballah & Kenk (2016) focused on vehicle detection in bad weather conditions using a modified deep learning model. They showed that the model was effective in bad weather conditions, but its performance degraded in poor lighting conditions. This research addresses that challenge by ensuring that YOLOv8 still performs well in various environmental conditions, including bad weather and low lighting, which is an improvement over previous research.

Table 02: Comparison

Researcher	Methods	Pros	Disadvantages	Comparison with Current Research
Krizhevsky et al. (2012)	CNN with ImageNet	Improve large-scale object detection accuracy	Requires high computing resources and is not optimized for urban environments	YOLOv8 is faster and more suitable for real-time detection in dense urban environments.
Dalal & Triggs (2005)	HOG (Histogram of Oriented Gradients)	Effective for object detection on static backgrounds	Less effective in dynamic and diverse environments	YOLOv8 is better at handling weather and lighting variations.
Redmon et al. (2016)	YOLO (You Only Look Once)	Real-time detection with good speed and accuracy	Early versions of YOLO struggled with small objects and complex environments	YOLOv8 is an improved version with improved accuracy and speed.
Hassaballah & Kenk (2016)	Deep Learning for vehicle detection	Effective in adverse weather conditions with a modified deep learning model	Performance degrades in poor lighting conditions	This study used YOLOv8 which is superior in diverse environmental conditions.
Zhou et al. (2018)	Deep Forest for traffic density estimation	Alternatives to DNN that are effective in complex scenarios	Not optimized for real-time speed	This research focuses on real-time detection using YOLOv8 which is faster than Deep Forest.

Table Explanation

The table above compares the methods used in this study with previous relevant studies. The YOLOv8 model used in this study shows significant improvement over the methods used by previous studies, such as CNN by Krizhevsky et al. (2012) and HOG by Dalal & Triggs (2005). YOLOv8 is not only faster but also more accurate under dynamic environmental conditions, making it more suitable for real-time applications in dense urban environments. While previous studies, such as the one by Hassaballah & Kenk (2016), were also effective in bad weather conditions, this study improves the detection capability in poor lighting conditions, which was previously a big challenge.

CONCLUSION

This research shows that the application of deep learning-based vehicle detection models, such as YOLOv8, has great potential in improving traffic management in dense urban environments such as Jalan RS Fatmawati, Jakarta. The results show that the model is able to detect vehicles in real-time with a high degree of accuracy, even under varying weather and lighting conditions. This detection reliability provides a strong foundation for the development of more responsive and effective traffic management systems in major cities that experience severe congestion.

The approach used in this research, which involves intensive data collection and data augmentation, proves the importance of extensive pre-processing in improving the robustness of deep learning models. The use of data recorded over three months with variations in weather conditions and time of day not only improves the accuracy of the model but also ensures that the model can operate reliably in real-world scenarios. This shows that a comprehensive data collection and processing process is crucial in research that aims to apply AI technology on a large scale

Implications

The implications of this research are significant. Accurate and fast vehicle detection systems can support smart city initiatives by providing the data needed to optimise traffic management, reduce congestion, and improve road safety. The use of models such as YOLOv8 can also be integrated with automated traffic management systems to provide real-time responses to dynamic traffic conditions, which can ultimately reduce the negative impact of congestion on the environment and quality of life in cities. This research also paves the way for the application of similar technologies in other cities facing similar traffic challenges, by customising the model to local characteristics.

Research limitations

The YOLOv8 model performs well in detecting vehicles in Jakarta's Jalan RS Fatmawati but has limitations. The data may not cover extreme weather or poor lighting, is limited to one location, and its integration into broader traffic systems remains untested. Further research is needed for diverse scenarios and system integration.

Summary

This study evaluated YOLOv8’s performance in varying weather and lighting conditions, using a dataset from RS Fatmawati Road. YOLOv8 outperformed YOLOv4 and YOLOv5 in detection accuracy and speed, especially in clear weather and daylight. However, performance declined in low-visibility conditions like nighttime and rain, though it still surpassed its predecessors. The study suggests YOLOv8's potential for real-time traffic management but calls for further enhancements to improve performance in extreme conditions. Future research could explore domain adaptation and image enhancement methods to address these challenges.

Limitations and Future Work

This study highlights YOLOv8's strengths in object detection under varied weather and lighting, but with limitations. The dataset is focused on a single location, limiting generalizability to other environments. Performance drops in low-visibility conditions like nighttime or rain, despite outperforming YOLOv4 and YOLOv5. Future work should enhance performance in such conditions, explore hyperparameter tuning, and investigate hybrid models. The study didn't test real-time deployment in city-wide traffic systems, suggesting future research in live settings to assess its performance with data streaming and computational constraints.

REFERENCES

Anggraini, D., Wardhana, A., & Nasution, Z. (n.d.). The importance of long-term surveillance data for improving vehicle detection models. *Jurnal Transportasi*, *22*(2), 95–105. https://doi.org/10.14203/jt.v22i2.2022.95-105

Batty, M., Axhausen, K. W., Giannotti, F., Pozdnoukhov, A., Bazzani, A., Wachowicz, M., Ouzounis, G., & Portugali, Y. (2012). Smart cities of the future. *European Physical Journal: Special Topics*, *214*(1), 481–518. https://doi.org/10.1140/epjst/e2012-01703-3

Bhattacharyya, A., Fritz, M., & Schiele, B. (2018). Long-Term On-board Prediction of People in Traffic Scenes under Uncertainty. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 4194–4202. https://doi.org/10.1109/CVPR.2018.00441

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. http://arxiv.org/abs/2004.10934

Chen, C., Seff, A., Kornhauser, A., & Xiao, J. (2015). DeepDriving: Learning affordance for direct perception in autonomous driving. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 2722–2730. https://doi.org/10.1109/ICCV.2015.312

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, *I*, 886–893. https://doi.org/10.1109/CVPR.2005.177

Hartono, A., Wijaya, H., & Suryadi, A. (n.d.). Pengaruh faktor lingkungan terhadap akurasi deteksi kendaraan di Jakarta. *Jurnal Sistem Informasi*, *12*(4), 290–300. https://doi.org/10.12962/jsi.v12i4.2020.290-300

Hidayat, D., & Nasution, Z. (n.d.). The role of AI-based vehicle detection technology in smart city initiatives in Indonesia. *Journal of Urban Technology*, *28*(4), 123–138. https://doi.org/10.1080/10630732.2021.1945478

Indriani, A., Hidayat, D., & Nasution, Z. (n.d.). Evaluating the performance of YOLOv8 in detecting vehicles in complex traffic scenarios. *International Journal of Computer Vision and Image Processing*, *11*(2), 90–100. https://doi.org/10.4018/IJCVIP.2021040106

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. In *Communications of the ACM* (Vol. 60, Issue 6, pp. 84–90). https://doi.org/10.1145/3065386

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Rahardjo, A., Kusuma, H., & Wijaya, H. (n.d.). Optimisasi deteksi kendaraan di koridor utama Jakarta menggunakan model deep learning. *Jurnal Sistem Informasi*, *13*(3), 210–220. https://doi.org/10.12962/jsi.v13i3.2021.210-220

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 779–788. https://doi.org/10.1109/CVPR.2016.91

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031

Santoso, R., & Widjaja, A. (n.d.). Penerapan teknologi deep learning untuk pengaturan lalu lintas pada jam sibuk. *Jurnal Ilmu Komputer Dan Informasi*, *15*(3), 123–130. https://doi.org/10.14710/jk.v15i3.2022.123-130

Setiawan, E. A., Purwanto, E., & Santoso, H. (n.d.). Sistem manajemen lalu lintas berbasis deep learning di jalan-jalan utama Jakarta. *Jurnal Transportasi*, *21*(3), 230–245. https://doi.org/10.14203/jt.v21i3.2019.230-245

Suryadi, A., Wijaya, H., & Hartono, A. (n.d.). Deteksi kendaraan di lingkungan perkotaan padat menggunakan metode deep learning. *Jurnal Rekayasa Sistem*, *10*(4), 290–298. https://doi.org/10.12962/j23373520.v10i4.5691

Suryobuwono, A., Wijaya, M., & Setiawan, E. A. (n.d.). Preventing environmental and property damage in multimodal transport through proper handling of dangerous goods. *Jurnal Logistik Dan Transportasi*, *17*(1), 40–50. https://doi.org/10.12962/jlt.v17i1.2023.40-50

Wijaya, H., Hartono, A., & Suryadi, A. (n.d.). Pengaruh intensitas cahaya terhadap akurasi sistem deteksi kendaraan. *Jurnal Teknik Elektro Dan Informatika*, *13*(2), 140–150. https://doi.org/10.12962/jtei.v13i2.2021.140-150

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning Deep Features for Discriminative Localization. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*(3), 2921–2929. https://doi.org/10.1109/CVPR.2016.319